Progress Report on QDP-JIT

F. T. Winter

Thomas Jefferson National Accelerator Facility

USQCD Software Meeting 2014
April 16-17, 2014 at Jefferson Lab
QDP-JIT/PTX provides a reimplemention of QDP++ for NVIDIA GPUs

- Automatic off-loading of expressions to the accelerators
- Multi-GPU support
- Dynamic PTX code generation
- Additional Just-In-Time (JIT) compilation step with NVIDIA driver
- Data layout is optimized for coalesced memory accesses
- Automatic H2D, D2H memory transfers via a 'software cache'
- Automatic tuning of CUDA kernels

- Paper accepted for publication in IEEE International Parallel & Distributed Processing Symposium 2014
QDP-JIT/LLVM Motivation

- Code maintainability
  - No template specializations (SSE, AVX, etc.) for each architecture
  - No heavy usage of `#ifdef` constructs

- Performance portability
  - Efficient code generation for all relevant targets
  - Not to be committed on compilers’ ability to deal with templated codes
  - Support for vector units, memory pre-fetchers, etc.
  - Efficient code: threading, scheduling, cache blocking, etc.

- Architecture independent implementation of QDP++

- LLVM is a framework worth targeting

- LLVM IR is architecture independent

- LLVM is embraced by HPC industry, e.g. NVIDIA, IBM, Intel, ...
QDP-JIT/PTX is limited to GPUs. To target a broader range of architectures a new LLVM IR code generator was implemented.
GPU route is still there via PTX, two approaches: The open source NVPTX backend or closed source libnvvm library.
- libnvvm part of CUDA since 5.5 and includes PTX-specific optimizations.
Generate x86 code with LLVM’s mature x86 backend. (Great SSE/AVX support)
Generate PowerPC 64 code. Some support for QPX (work in progress).
New architectures supported provided that it supports JIT compilation.
QDP++ specifies the data layout through the nesting order of templated data types:
Outer < Spin < Color < Reality < float > > >

QDP-JIT splits the outer loop by an optional inner vector length \( I \)
Outer < Spin < Color < Reality < Inner < float > > > >

The code generation step “intercepts and changes the data layout”
Spin < Color < Reality < Outer < Inner < float > > > > > (GPUs, \( I = 1 \))
Outer < Spin < Color < Reality < Inner < float > > > > > (CPUs with SSE/AVX, \( I = 2/4/8 \))
Outer < Spin < Color < Inner < Reality < float > > > > > (BG/Q, \( I = 2 \) (DP))
Benchmark on Intel Sandy Bridge

- Out of L2 cache for local problem sizes larger than $L = 20^4$.
- Within cache the code achieves up to 78% peak of E5-2650 at 2.00GHz, 256 GFlops (SP) peak.
Benchmark on Intel Sandy Bridge

- Out of L2 cache for local problem sizes larger than $L = 16^4$.
- Within cache the code achieves up to 78% peak of E5-2650 at 2.00GHz, 128 GFlops (DP) peak
QPX instructions are generated, there are however still alignment issue
Out of L2 cache for local problem sizes larger than $L = 16^4$. 
- GCC on vanilla QDP++ is currently doing better on the linear algebra than QDP-JIT/LLVM.
- Mainly because the LLVM BG/Q backend misses essential performance features.
Benchmark on Blue Gene/Q, preliminary

- Shifting of sub-lattices
- Overlapping of computation and off-node communication.
- For rb2 Wilson DSlash preliminary measurements show a speedup factor of $\times 12.4$. 

![Graph showing performance in GFlops against BG/Q nodes for QDP-JIT and QDP++, with speedup factor $\times 12.4$.]
QDP-JIT/LLVM provides an architecture independent implementation of QDP++

Runs Chroma HMC (Wilson Clover) on GPUs, x86, and BG/Q

Optimizations:
- Custom data layout to support vectorization
- Multi-threading
- Sub-lattice shifting
- Overlapping MPI and compute

Improve performance on BG/Q (QPX, SPI)

Intel Xeon Phi (KNL)

Apply advanced optimizations:
- Polyhedral model
- Cache blocking
- Memory prefetching